

A Note on NP -Hardness of Preemptive Mean Flow-Time

Scheduling for Parallel Machines

Odile Bellenguez-Morineau · Marek Chrobak · Christoph

Dürr · Damien Prot

Received: date / Accepted: date

Abstract In the paper “*The complexity of mean flow time scheduling problems with release times*”, by Baptiste, Brucker, Chrobak, Dürr, Kravchenko and Sourd, the authors claimed to prove strong NP -hardness of the scheduling problem $P|pmtn, r_j| \sum C_j$, namely multiprocessor preemptive scheduling where the objective is to minimize the mean flow time. We point out a serious error in their proof and give a new proof of strong NP -hardness for this problem.

Keywords Scheduling · Complexity · Identical machines · Preemptive problems

1 Introduction

In Baptiste et al. [2007] the following scheduling problem was considered. We are given N jobs, where each job j has some release time r_j and processing time p_j , all positive integers. The goal is to preemptively schedule these

Odile Bellenguez-Morineau · Damien Prot

LUNAM Université, École des Mines de Nantes, IRCCyN UMR CNRS 6597 (Institut de Recherche en Communication et en Cybernétique de Nantes), 4 rue Alfred Kastler, La Chantrerie, BP20722, 44307 Nantes Cedex 3, France

E-mail: odile.morineau, damien.prot@mines-nantes.fr

Marek Chrobak

Computer Science Department, University of California, Riverside, CA 92521, USA

E-mail: marek@cs.ucr.edu

Christoph Dürr

CNRS, LIP6, Université Pierre et Marie Curie, 4 place Jussieu, 75252 Paris Cedex 05, France

E-mail: Christoph.Durr@lip6.fr

jobs on m parallel identical machines, so as to minimize the mean flow time (or, equivalently, the total completion time). We use a standard definition of preemptive schedules, namely a schedule is specified by assigning to each job j a finite set of execution intervals of j , where each such interval is associated with some machine k . For a schedule to be feasible, all intervals associated with any machine k must be disjoint, and all intervals assigned to the same job j must be disjoint and start not earlier than at time r_j . The completion time of a job j , denoted by C_j , is defined as the right endpoint of the last execution interval assigned to j .

In the three-field notation for scheduling problems, this problem is denoted $P|pmtn, r_j|\sum C_j$. The special case when there are only $m = 2$ parallel machines has been shown to be NP -hard in Du et al. [1990], while the single machine variant is solvable in polynomial time, see Baker [1974].

The computational complexity of $P|pmtn, r_j|\sum C_j$ was studied by Baptiste et al. [2007], and one of the results in that paper was a proof of strong NP -hardness. Unfortunately, as we show in the next section, that proof has a serious flaw. Therefore we provide a new proof of strong NP -hardness in Section 3, which builds on the overall structure of the proof from Brucker and Kravchenko [2004].

2 Counterexample to the Proof in Baptiste et al. [2007]

The (faulty) strong NP -hardness proof of Baptiste et al. [2007] is based on a reduction from 3-PARTITION. It converts an instance of 3-PARTITION into a collection of jobs of three types: x-jobs, B-jobs and 1-jobs. The role of the B- and 1-jobs is to force any optimal schedule to start these jobs at their release time, and leave only a small time interval available for the x-jobs. The key idea of the proof was that in an optimal schedule, the x-jobs would have to be scheduled without preemption in this interval, with three jobs per machine, and in this way the resulting schedule would represent a solution of the original instance of 3-PARTITION.

The error in this proof is that it is always possible to schedule the x-jobs in the allowed interval using the method described by McNaughton [1959] (see Theorem 3.1): Order the jobs $1, 2, \dots, N$ arbitrarily. Using the unit slots of the jobs, in this order, fill the allowed interval for machine 1, going from left to right, then fill the allowed interval for machine 2, and so on, until all jobs are processed.

We now show a specific counter-example to the construction of Baptiste et al. [2007]. Let the instance of 3-PARTITION consist of positive integers x_1, \dots, x_{3n}, y that satisfy $\sum_{i=1}^{3n} x_i = ny$ and $\frac{y}{4} < x_i < \frac{y}{2}$ for each i . The objective is to decide if there exist a partition of $\{1, \dots, 3n\}$ into n sets P_1, \dots, P_n such that $\sum_{i \in P_k} x_i = y$ for all k .

Let $A = 6ny$ and $B = 18n^2y^2$. The reduction in Baptiste et al. [2007] converts the above instance of 3-PARTITION into an instance of $P|pmtn, r_j| \sum C_j$ with n machines and $N = 4n + An$ jobs of three types:

- x-jobs j : for all $j \in \{1, \dots, 3n\}$, with $r_j = 0$ and $p_j = Ax_j$,
- B-jobs j : for all $j \in \{3n + 1, \dots, 4n\}$, with $r_j = Ay$ and $p_j = B$,
- 1-jobs j : for all $j \in \{4n + 1, \dots, N\}$, with $r_j = Ay + B$ and $p_j = 1$.

The authors claim that the instance of 3-PARTITION has a solution if and only if there exists a schedule with $\sum_{j=1}^N C_j \leq D$, where $D = 3nAy + n(Ay + B) + n \sum_{i=1}^A (Ay + B + i)$. The “only if” part of this claim is easy: on a machine k , we first schedule the three x-jobs $j \in P_k$, then one B-job and finally A 1-jobs. As the example below shows, however, the “if” implication is not valid.

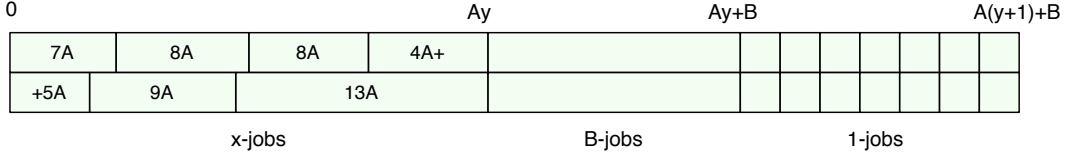


Fig. 1 The schedule of the jobs from our counter-example. (Picture is not to scale.) The x-jobs are scheduled using McNaughton’s algorithm. The 4th x-job is scheduled in two parts, one of length $4A$ scheduled on machine 1 and the other, of length $5A$, on machine 2.

Our counter-example is an instance of 3-PARTITION with $n = 2$ and $x_1 = 7$, $x_2 = 8$, $x_3 = 8$, $x_4 = 9$, $x_5 = 9$, $x_6 = 13$ and $y = 27$. Clearly, there is no solution to 3-PARTITION since the partition set containing x_6 would need two additional numbers that add up to $27 - 13 = 14$, which is not possible. In the corresponding instance of $P|pmtn, r_j| \sum C_j$ we will have $A = 324$ and $B = 52488$. The instance consists of $n = 2$ machines and 656 jobs:

- Six x-jobs with processing times $7A$, $8A$, $8A$, $9A$, $9A$ and $13A$, all released at time 0,
- Two B-jobs with processing time B , released at time $27A$, and
- $2A$ 1-jobs with processing time 1, released at time $27A + B$.

As shown in Figure 1, all x-jobs in this instance can be scheduled in the time interval $[0, 27A]$ using McNaughton’s method, and the objective value of the shown schedule is at most $D = 3nAy + n(Ay + B) + n \sum_{i=1}^A (Ay + B + i)$, because all x-jobs complete no later than at time Ay . Thus the “if” implication does not hold.

3 A New Proof of \mathbb{NP} -Hardness

In this section we present a corrected proof. Our proof, as before, uses a reduction from 3-PARTITION, although the construction is more involved.

It has been proven that, for any instance of $P|pmtn, r_j| \sum C_j$, there exists an optimal schedule where pre-emptions occur only at integer times (see Baptiste et al. [2007], for example); hence we will make this assumption throughout the paper, namely we will assume that the time is divided into unit time slots, each either idle or fully filled by a unit fragment of one job.

Let us fix an instance of 3-PARTITION consisting of positive integers x_1, \dots, x_{3n}, y , where $\sum_i x_i = ny$ and $y/4 < x_i < y/2$ for all i . Without loss of generality, we assume that $n \geq 2$. Using this instance, we construct an instance of $P|pmtn, r_j| \sum C_j$ with n machines. In this construction we use the following values:

$$\begin{aligned}\lambda &= 2n(n-1) \\ L &= n\lambda y + \lambda \\ \text{OPT} &= yL \frac{n(n-1)}{2} + nyL \frac{n(n-1)}{2} + n\lambda y \\ T &= n^2 yL + 3n\text{OPT}\end{aligned}$$

Let $B_i = \sum_{l=1}^{i-1} (nLx_l + \text{OPT})$ for $i = 1, \dots, 3n$. Note that $B_1 = 0$. For convenience, we also let $B_{3n+1} = T$. We partition the time interval $[0, T)$ into $3n$ blocks, where the i -th block is the interval $[B_i, B_{i+1})$. Thus each block i , has length $nLx_i + \text{OPT}$.

We also have another special “cork” block in the interval $[T, T + \text{OPT})$. The instance will have jobs of four types:

S-jobs: In each block i , for each integer time $t = B_i, B_i + 1, \dots, B_i + (n-1)Lx_i - 1$, we release $n - \lceil \frac{t-B_i+1}{Lx_i} \rceil$

jobs with unit processing time. The idea is that these S-jobs should form a staircase-shaped schedule in their block i , with each stair step having (ideally) length Lx_i .

X-jobs: These jobs correspond to the numbers x_i in the original 3-PARTITION instance: in each block i , we will have one job X_i of length λx_i released at the beginning of the block, i.e. at time B_i .

F-jobs: This is a set of n jobs F_1, \dots, F_n , released at time 0. Their role is to fill the idle times in each machine k . The length of each job F_k is $T - (k-1)Lny - \lambda y$.

C-jobs: This is a set of n_{OPT} unit jobs released in the cork block. Specifically, for each time $t = T, \dots, T + \text{OPT} - 1$, we release n unit jobs at time t . The purpose of these jobs is to force all S-jobs, X-jobs and F-jobs to complete no later than at time T .

Note that this transformation can be computed in polynomial time if the instance of 3-PARTITION and the constructed instance of $P|pmt_n, r_j| \sum C_j$ are represented in the unary encoding. This will be sufficient for our purpose, since 3-PARTITION is strongly NP-hard.

To simplify calculations, instead of minimizing $\sum C_j$, we will use the objective function $\sum D_j$ where $D_j = C_j - r_j - p_j$, which is of course equivalent. We will refer to D_j as the *delay* of job j . For a schedule σ , by $\text{delay}(\sigma)$ we will denote the total delay (that is, $\sum D_j$) of σ . For $\phi \in \{S, X, F, C\}$, by $\text{delay}_\phi(\sigma)$ we denote the contribution of ϕ -jobs to the total delay in σ .

Theorem 1 *The instance of 3-PARTITION has a solution if and only if the instance of $P|pmt_n, r_j| \sum C_j$ constructed above has a schedule with total delay at most OPT.*

The rest of this section is devoted to the proof of Theorem 1. We will prove the two implications in the theorem separately.

(\Rightarrow) For the “only if” direction, consider a partition P_1, \dots, P_n such that $\sum_{i \in P_k} x_i = y$ for every k . Schedule all S-jobs at their release times to form stairs in each block i . Schedule the C-jobs at their release times, to form the cork block. For each k , if $i \in P_k$, then schedule the X-job corresponding to x_i at offset $(k-1)Lx_i$ in block i on machine k . These jobs are scheduled without preemption. For each k , schedule job F_k preemptively on machine k , so that it completes at time T . By the property of the sets P_1, \dots, P_n , all n machines are now completely filled up to time $T + \text{OPT}$. Figure 2 shows an illustration of such a schedule.

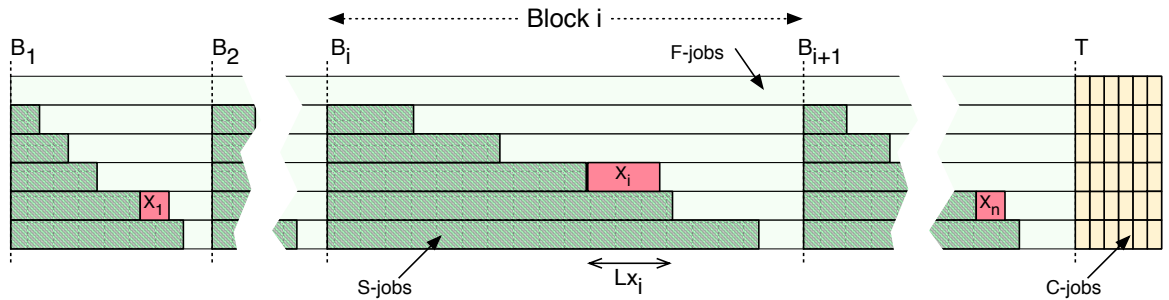


Fig. 2 A schedule with delay equal to OPT.

We now focus on the delay of this schedule. The S-jobs and C-jobs do not contribute to the delay, as they start at their release time. The F-jobs complete at time T and hence generate a delay of $nyL \frac{n(n-1)}{2} + \lambda ny$. Finally, all X-jobs that complete on a machine k generate a total delay of $(k-1)Ly$. Adding up the delays of different types of jobs, we obtain the total delay of OPT, as required.

(\Leftarrow) In the following, the “if” direction of the proof is detailed. As a first step, we fix a schedule σ of delay at most OPT. To simplify the argument, we first argue that we can make some simplifying assumptions about σ .

We first assume that σ is OPT-dominant, where the dominance property is defined as follows. For two schedules σ_1, σ_2 , we say that σ_1 *strictly dominates* σ_2 if either

- (i) there is a time t_0 such that in all time slots $t < t_0$, σ_1 and σ_2 schedule the same number of S-jobs, while at time t_0 , σ_1 schedules strictly more S-jobs than σ_2 , or
- (ii) σ_1 and σ_2 execute the same number of S-jobs in each time slot, and there is a time t_0 such that in all time slots $t < t_0$, σ_1 and σ_2 schedule the same number of X-jobs, while at time t_0 , σ_1 schedules strictly more X-jobs than σ_2 . (As we shall prove shortly, we can assume that at most one X-job is executed at any time.)

Then σ is called OPT-dominant if it is not strictly dominated by any other schedule of delay at most OPT. Clearly, there may be many OPT-dominant schedules.

In addition we assume the *vertical ordering property*, stating that in every time slot, jobs are sorted from the first to the last machine according to the order $F_1, \dots, F_n, X_1, \dots, X_{3n}$, followed by S-jobs and then followed by C-jobs. Reordering the units of jobs scheduled at the same time slot on different machines has no impact on the delay of the schedule, so this last assumption can be made without loss of generality.

Lemma 1 *In σ , all X-jobs and all S-jobs complete strictly before the end of their blocks, and the F-jobs complete no later than at time T .*

Proof Since the total delay is assumed to be at most OPT, and no jobs are released in the last slots of each block, all S-jobs and X-jobs have to complete strictly before the end of their corresponding blocks.

To show the second claim, without loss of generality, we assume that C-jobs are scheduled in order of release times, that is, for any two C-jobs i and j , $r_i < r_j$ implies that j is scheduled not earlier than i . Suppose that some F-job completes at time $T + \tau$, where $\tau > 0$. If $\tau \geq \text{OPT} + 1$, then the delay of this F-job exceeds OPT. Otherwise, using our assumption about C-jobs, this F-job forces a delay for at least one C-job released in each time slot of the cork block. Since the F-job has itself a delay of at least τ , so the total delay would also exceed OPT. \square

Lemma 2 *There is no idle time in σ before time T .*

Proof Using Lemma 1, all S-jobs, X-jobs and F-jobs have to be finished at time T . Since the total processing time of these jobs is nT , the lemma follows. \square

Lemma 3 *In σ , all F-jobs complete exactly at time T .*

Proof At time $T - 1$, all S-jobs and X-jobs are already completed, by Lemma 1. So only F-jobs can execute at time $T - 1$. The lemma now follows from Lemma 2. \square

As previously observed, we can assume that the C-jobs do not generate any delay. By Lemma 3, the total delay of F-jobs is $nyL \frac{n(n-1)}{2} + \lambda ny$. Removing this contribution of F-jobs to the objective function leads to the following bound, which will play an important role in the rest of the proof:

$$\text{delay}_S(\sigma) + \text{delay}_X(\sigma) \leq yL \frac{n(n-1)}{2}. \quad (1)$$

The lemma below says that the S-jobs of each block must be scheduled in σ so as to form a staircase shape, similar (but not necessarily identical) to the schedule shown in Figure 2.

Lemma 4 *Schedule σ has the following property: in each block, going from left to right, the numbers of S-jobs in the time slots of this block form a non-increasing sequence.*

Proof The proof uses an argument by contradiction. Suppose that inside some block there are time slots $t - 1$ and t such that at t there are strictly more S-jobs scheduled than at $t - 1$. For the purpose of this proof, we may assume that S-jobs are scheduled in order of their release times, since delay_S is independent of the ordering of the S-jobs. Therefore one of the S-jobs from time slot t must be already released at time $t - 1$, by the release pattern of the S-jobs. Call this job ℓ . At time $t - 1$ no machine can be idle, by Lemma 2. So at time $t - 1$ we have more units of X- or F- jobs than at time t . This implies that there is a job ℓ' of type X or F scheduled at time $t - 1$ that is not scheduled at time t . Hence we can exchange these units of ℓ and ℓ' without increasing the total delay. But this contradicts the assumption that σ is OPT-dominant. \square

Lemma 5 *Each machine, within each block, executes units of jobs in the following order: first S-jobs, then X-jobs (if any), and finally F-jobs.*

Proof Fix some block i . By Lemma 4, the slots occupied by S-jobs in this block form a staircase shape; thus on each machine they are executed before X-jobs and F-jobs. We still need to show that slots occupied by X-jobs precede those occupied by F-jobs. We know that only one X-job is executed in this block, namely X_i .

We argue by contradiction. Suppose that there is a time t in this block such that some machine k executes X_i at time t and executes some F-job at time $t - 1$. At both times, according to vertical ordering property of σ , all machines $1, \dots, k - 1$ execute F-jobs and machines $k + 1, \dots, n$ execute S-jobs. This implies that there is some F-job executed at time $t - 1$ that is not executed at time t . We can then exchange this F-slot with the slot of X_i at time t , without increasing the delay (by Lemma 3), thus obtaining a contradiction with OPT-dominance of σ .

□

Lemmas 1 to 5 characterize the structure of our schedule σ :

- i. All C-jobs are scheduled at their release times.
- ii. All S-jobs and X-jobs complete in their respective blocks.
- iii. The F-jobs fill in the remaining slots in the blocks and they complete exactly at time T .
- iv. In each block i , for each k , the number of S-jobs executed by machine k is non-decreasing with k ; that is, the S-jobs form a staircase pattern in each block.
- v. In each block i , each machine k first executes some S-jobs, then some number of units of X_i (if any), and then some units of F-jobs.

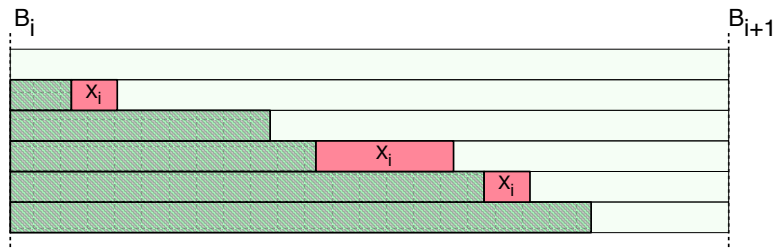


Fig. 3 The structure of a block in schedule σ .

Thus the overall structure of σ is similar to the schedule shown in Figure 2; however, within each block i the steps of the staircases of S-jobs may have different lengths, and X_i may not be scheduled in one contiguous block on one machine. A typical structure of a block is shown in Figure 3.

For each machine k and each block i , the *ideal number* of S-jobs on machine k in block i is defined to be $(k-1)Lx_i$. This value corresponds to the number of S-jobs processed on machine k in block i in the (\Rightarrow) direction of the proof (see Figure 2). We also define E_k^i to be the *excess* of S-jobs on machine k in block i , namely the difference between the actual and the ideal number of S-jobs on machine k in block i , if this difference is non-negative; otherwise let E_k^i be 0. More specifically, if the number of S-jobs scheduled by σ on machine k in block i is c then $E_k^i = \max\{c - (k-1)Lx_i, 0\}$.

Lemma 6 *In schedule σ , the total delay of the S-jobs is bounded by the following inequality:*

$$\text{delay}_S(\sigma) \geq \sum_i \sum_k Lx_i E_k^i.$$

Proof For every time t , let $z(t)$ be the difference between the number of S-jobs released at times $0, 1, \dots, t$ and the number of S-jobs scheduled at the same times in σ . Clearly we have $z(t) \geq 0$ for every time t .

Consider an S-job j with release time r_j and completion time C_j . Its delay is $D_j = C_j - 1 - r_j$. Job j contributes one unit to $z(t)$ for each time $t = r_j, r_j + 1, \dots, C_j - 2$; hence $\text{delay}_S(\sigma) = \sum_t z(t)$.

Now fix a block i and machine $k \geq 2$ with $E_k^i > 0$, and let $t_k^i = B_i + (k-1)Lx_i$. In the interval $[t_k^i, t_k^i + E_k^i)$ at least $(n-k+1)E_k^i$ S-jobs are scheduled, but only $(n-k)E_k^i$ S-jobs are released. The additional scheduled S-jobs must therefore be released strictly before time t_k^i , which implies $z(t_k^i - 1) \geq E_k^i$.

At each time $\tau = t_k^i - Lx_i, \dots, t_k^i - 1$ the number of released S-jobs is $n-k+1$. By the choice of i and k , and by Lemma 4, at each time τ in this range the number of executed S-jobs is at least $n-k+1$. We thus obtain that $z(\tau)$ is non-increasing in this range, so we have

$$z(t_k^i - Lx_i) \geq \dots \geq z(t_k^i - 1) \geq E_k^i,$$

and thus

$$z(t_k^i - Lx_i) + \dots + z(t_k^i - 1) \geq Lx_i E_k^i,$$

which concludes the proof of the lemma by summing over all choices of i and k . \square

We complete the proof with an upper bound on the excess values, that motivates our choice for the value of parameter λ .

Lemma 7 *We have $\sum_i \sum_k E_k^i < \lambda$.*

Proof For a proof by contradiction assume that the left hand side is at least λ . Then Lemma 6, together with the assumption that $x_i > y/4$ for all i , would imply that

$$\text{delay}_S(\sigma) > \frac{yL}{4} \sum_i \sum_k E_k^i \geq \frac{yL}{4} \lambda = yLn(n-1)/2.$$

This, however, contradicts the bound in Equation (1).

□

We denote by P_k the set of indices of the X-jobs completing on a machine k , independent of whether they are scheduled entirely on machine k or not. By the construction, the lengths of all X-jobs are multiples of λ , which is a strict upper bound on the number of S-jobs that can diverge from the ideal pattern (by Lemma 6). We will use this fact, to show that the partition P_1, \dots, P_n forms a solution to the original instance of 3-PARTITION.

Lemma 8 *The following inequality holds for all machines k :*

$$\sum_{i \in P_k \cup \dots \cup P_n} x_i \geq (n - k + 1)y.$$

Proof The amount of S-jobs executed by each machine ℓ can be bounded by the amount of S-jobs in the ideal pattern (from the proof of the “if” implication) plus the excess values for all blocks, which works out to be at most $(\ell - 1)Lny + \sum_i E_\ell^i$. Therefore, by Lemma 7, the total amount of S-jobs executed on machines numbered k, \dots, n is strictly smaller than

$$(k - 1)Lny + \dots + (n - 1)Lny + \lambda.$$

By the vertical ordering assumption, each job F_ℓ can only be scheduled on machines $1, \dots, \ell$ and therefore the total amount of F-jobs on machines numbered k, \dots, n is at most

$$[T - (k - 1)Lny - \lambda y] + \dots + [T - (n - 1)Lny - \lambda y].$$

Together, the total amount of S-jobs and F-jobs on machines k, \dots, n is strictly smaller than $(n - k + 1)T - (n - k + 1)\lambda y + \lambda$. Thus, using Lemma 2, the total length of X-jobs completing on machines k, \dots, n satisfies

$$\sum_{i \in P_k \cup \dots \cup P_n} x_i > (n - k + 1)\lambda y - \lambda.$$

The lemma follows from the fact that every X-job has length that is a multiple of λ . □

To complete the proof of the theorem, we focus on the delay of the X-jobs.

We know that if some machine k executes at least one unit of a job X_i in block i , then all earlier slots of machine m in this block execute S-jobs. Therefore the amount of S-jobs on the machines and in blocks where

X-jobs complete can give us a lower bound on the total completion time of the X-jobs, from which we have to subtract the total processing time to obtain a bound on $\text{delay}_X(\sigma)$.

In the ideal schedule, the delay of X-jobs that complete on a machine k is $(k-1)Ly$, but in σ this delay could be different. To get a lower bound on this delay, define the *deficiency of S-jobs on machine k in a block i* to be the ideal value of $(k-1)Lx_i$ minus the number of S-jobs executed by k in block i of schedule σ , if this value is non-negative; otherwise we let the deficiency to be 0. Then the delay of X-jobs that complete on a machine k is at least $(k-1)L \sum_{i \in P_k} x_i$ minus the total deficiency of S-jobs on machine k . But the total deficiency of S-jobs, overall all machines and all blocks, is the same as the total excess $\sum_i \sum_k E_k^i$, so it is strictly smaller than λ , by Lemma 7. This leads to the lower bound

$$\text{delay}_X(\sigma) > L \sum_{i \in P_2} x_i + 2L \sum_{i \in P_3} x_i + \dots + (n-1)L \sum_{i \in P_n} x_i - \lambda - n\lambda y, \quad (2)$$

where we use the fact that the total processing time of the X-jobs is $n\lambda y$.

Putting everything together now, we apply Inequality (1), Inequality (2) above, substitute $L = ny\lambda + \lambda$, and then apply Lemma 8 (summing over all values of k), obtaining the following sequence of inequalities:

$$\begin{aligned} yL \frac{n(n-1)}{2} &\geq \text{delay}_X(\sigma) \\ &> L \left(\sum_{i \in P_2 \cup \dots \cup P_n} x_i + \sum_{i \in P_3 \cup \dots \cup P_n} x_i + \dots + \sum_{i \in P_n} x_i \right) - L \\ &\geq yL \frac{n(n-1)}{2} - L. \end{aligned}$$

The first and last expressions are multiples of L . From this derivation we can thus conclude that the bound from Lemma 8, used in the last inequality, must be in fact tight. Specifically, we get that $\sum_{i \in P_k \cup \dots \cup P_n} x_i = (n-k+1)y$ for each $k = 2, \dots, n$. We thus obtain that $\sum_{i \in P_1} x_i = y$ and, proceeding by induction on k , we have $\sum_{i \in P_k} x_i = y$ as well for all other machines k . This completes the proof of Theorem 1.

Acknowledgements. M. Chrobak was partially supported by National Science Foundation grant CCF-1217314.

The authors would like to thank anonymous reviewers for many useful comments and suggestions.

References

- K.R. Baker. *Introduction to Sequencing and Scheduling*. John Wiley & Sons, New York, 1974.
- P. Baptiste, P. Brucker, M. Chrobak, C. Dürr, S.A. Kravchenko, and F. Sourd. The complexity of mean flow time scheduling problems with release times. *Journal of Scheduling*, 10:139–146, 2007.

- P. Brucker and S.A. Kravchenko. Complexity of mean flow time scheduling problems with release dates. Technical report, Universität Osnabrück, Fachbereich Mathematik/Informatik, OSM Reihe P, Heft 251, 2004.
- J. Du, J.Y.-T. Leung, and G.H. Young. Minimizing mean flow time with release time constraint. *Theoretical Computer Science*, 75:347–355, 1990.
- R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6:1–12, 1959.